

EXHIBIT 2

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

ARISTA NETWORKS, INC.
Petitioner

v.

CISCO SYSTEMS, INC.
Patent Owner

Case IPR2016-00244
Patent 7,953,886

PATENT OWNER PRELIMINARY RESPONSE

Mail Stop PATENT BOARD
Patent Trial and Appeal Board
U.S. Patent & Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

Table of Contents

I.	Introduction.....	1
II.	The '886 patent provides a comprehensive solution to simplifying interaction with a router's command line interface.....	2
III.	Claim construction.....	7
IV.	Ground 1: Petitioner failed to show that claims 1–10 are rendered obvious by Gorthy, Froyd, Courtney, and JUNOScript.....	8
A.	Overview of the applied references.....	8
1.	Gorthy's configuration schema-based system.	8
2.	Froyd's generalized markup.	12
3.	Courtney's network device configuration modeling system. ...	13
4.	JUNOScript's router API.....	16
B.	Petitioner failed to show that the combination of Gorthy, Froyd, Courtney, and JUNOScript discloses the “translating an output message...” element of the challenged claims.....	19
1.	Courtney's XML router configuration is not the claimed “output message.”	20
2.	JUNOScript does not teach or suggest the claimed “output message.”	24
C.	Petitioner has failed to establish that a person of ordinary skill in the art would combine Gorthy, Courtney, Froyd, and JUNOScript.	32
D.	Petitioner did not establish that JUNOScript is a § 102 prior art printed publication.....	36
V.	Conclusion.	38

Table of Authorities

<i>Blue Calypso LLC v. Groupon Inc.</i> , Case. No., 2015-1396, (Fed Cir., March 1, 2016)	36
<i>Bumble Bee Foods, LLC v. Kowalski</i> , Case IPR2014-00224, Paper 18 (June 5, 2014)	33
<i>Google Inc. v. EveryMD.com LLC</i> , IPR2014-00347, Paper 9 (May 22, 2014)	33
<i>General Electric Co. v. TAS Energy Inc.</i> , IPR2014-00163, Paper 11 (May 13, 2014)	33
<i>Hilgraeve, Inc. v. Symantec Corp.</i> , 271 F.Supp.2d 964 (E.D. Mich. 2003)	38
<i>In re Kahn</i> , 441 F.3d 977 (Fed. Cir. 2006)	26, 34
<i>In re Lister</i> , 583 F.3d 1307 (Fed. Cir. 2009)	37
<i>Johnson Controls, Inc. v. Wildcat Licensing WI, LLC</i> , IPR2014-00305, Paper 45 (June 22, 2015)	8
<i>Kinetic Techs., Inc. v. Skyworks Solutions, Inc.</i> , IPR2014-00529, Paper 8 (Sept. 23, 2014)	26, 31, 35, 36
<i>KSR Int’l v. Teleflex Inc.</i> , 550 U.S. 398 (2007)	32–33
<i>Mannesmann Demag Corp. v. Engineered Metal Products Co., Inc.</i> , 605 F. Supp. 1362 (D. Del. 1985), aff’d, 793 F.2d 1279 (Fed. Cir. 1986)	37
<i>ManukaMed Ltd. v. Apimed Medical Honey Ltd.</i> , IPR2013-00234, Paper 17 (Sept. 25, 2013)	33

IPR2016-00244
U.S. Patent No. 7,953,886

<i>Medtronic, Inc. v. Robert Bosch Healthcare Sys.</i> , Case IPR2014-00436, Paper 17 (June 19, 2014).....	33
<i>Motivepower, Inc. v. Cutsforth, Inc.</i> , IPR2013-00267, Paper 10 (Nov. 1, 2013)	8
<i>Neutrino Dev't Corp. v. Sonosite Inc.</i> , 337 F.Supp.2d 942 (S.D. Tex. 2004), aff'd, 210 Fed. Appx. 991 (Fed. Cir. 2006)	37
<i>SRI Int'l v. Internet Sec. Sys.</i> , 511 F. 3d 1186 (Fed. Cir. 2008).....	36
<i>TRW Auto US LLC. v. Magna Elecs. Inc.</i> , Case IPR2014-00296, Paper 15 (July 3, 2014).....	33
<i>Unigene Labs., Inc. v. Apotex, Inc.</i> , 655 F.3d 1352 (Fed. Cir. 2011).....	33, 34
<i>Valeo Inc., et al. v. Magna Elec., Inc.</i> , IPR2014-00222, Paper 13 at 17 (Apr. 23, 2015)	37–38
<i>Wangs Alliance Corp. v. Koninklijke Philips N.V.</i> , IPR2015-01287, Paper 8 (Nov. 25, 2015)	8

IPR2016-00244
U.S. Patent No. 7,953,886

Exhibit List

Exhibit No.	Description
2001	Cisco Networking for Dummies, 2 nd Edition (2002)
2002	Juniper MX Series, A Comprehensive Guide to Trio Technologies on the MX (2012)
2003	JUNOS OS for Dummies, 2 nd Edition (2008)
2004	JunoScript API Reference, Release 5.1 (2001)

I. Introduction.

Prior to the '886 patent, router manufacturers faced a dilemma. Users were familiar with the existing command line interfaces (CLIs) available on a router. CLIs were human-machine interfaces, meaning generally they allow a human to interact with a router. But generally, these CLIs were not suitable to be machine-machine interfaces for purposes of automation. The '886 patent addressed this predicament through a novel router configuration and management interface that supports both extensible markup language (XML) and original CLI commands. By translating both input and output, the '886 patent discloses a system that can act as a comprehensive interface for automation, while still also allowing for backward compatibility with CLI commands because the CLI is still capable of accepting the original CLI commands.

The Petition presents a single ground of rejection, relying on a combination of four references—Gorthy, Froyd, Courtney, and JUNOScript—to render obvious claims 1–10 of the '886 patent. But, the Board should deny this Petition because none of these four references, either alone or in any combination, teach or suggest the elements of the claims. Specifically, the proposed combination at least does not teach or suggest converting into XML a CLI output generated in response to execution of an XML-formatted input command.

The Board should also deny the Petition for another reason. Petitioner's cursory treatment of its motivation to combine Gorthy, Froyd, Courtney, and JUNOScript is insufficient to establish a *prima facie* case of obviousness. Petitioner's argument lacks the requisite rational underpinning because it only makes generic, conclusory statements regarding the applied references. And, Petitioner's expert does not salvage Petitioner's argument because Dr. Clark merely repeats the same reasoning verbatim without providing any supporting evidence or expert insights.

For both of these reasons, Petitioner's sole challenge to claims 1–10 fails for all claims, and the Petition should be denied.

II. The '886 patent provides a comprehensive solution to simplifying interaction with a router's command line interface.

The '886 patent is directed to providing a comprehensive extensible markup language interface for monitoring and configuring a router, while still maintaining the CLI. Specifically, in the disclosed embodiment, the '886 patent solution translates XML input to CLI input and CLI output to XML output. XML, as an example of "extensible markup language," is a method of describing data by adding identifiers to indicate logical components or layout of the data.

Historically, a popular method for configuring and monitoring a router was through the use of a CLI. A CLI allows users to "send[] commands and

instructions to and receive information from the router itself.” (Exhibit 1001, ’886 patent, 1:12–15.) In other words, the CLI functions as both an input and an output system to a router. One CLI is used by an exemplary internetwork operating system called the IOS operating system (IOS CLI). (’886 patent, 1:15–16.) The IOS CLI is a “comprehensive interface, which has expanded continuously as technology has improved over the past twenty years.” (’886 patent, 1:16–17.)

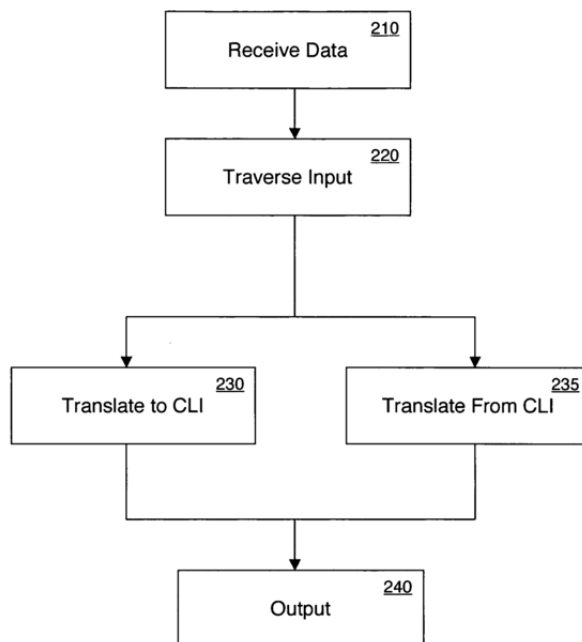
At the time of the invention many users had developed “complicated scripts to handle various configuration and access needs.” (’886 patent, 1:21–22.) These scripts often rely on a router’s CLI supporting particular input commands and outputting information in a particular format so that the script can properly parse the information. As router software is upgraded and the CLI changes, a user’s scripts could become obsolete if the command format relied upon in the script changed.

The CLI interfaces were also not originally developed with an eye towards automation. (’886 patent, 1:30–34.) For example, the format of CLI output was not uniform, and thus was difficult to process automatically. (*Id.*) Accordingly, this meant that the commands in CLIs were “a very difficult and cumbersome task to automate.” (’886 patent, 1:32–33.)

The ’886 patent addressed these problems by providing the same CLI commands that users were accustomed to while also “allow[ing] for an easy, more

structured approach to accessing and configuring a router.” (’886 patent, 1:34–38.) Specifically, in one embodiment, the novel system of the ’886 patent provides an “IOS/CLI Parser 110” that can handle both commands formatted according to normal “CLI rules and behaviors” and “in accordance with an XML schema of CLI rules and behaviors.” (’886 patent, 3:22–29.) In other words, the novel CLI Parser of the ’886 patent can either receive normal CLI commands or XML-formatted CLI commands. The CLI parser in this embodiment is also capable of outputting CLI responses “into an XML response in accordance with an XML schema of the CLI rules and behaviors.” (’886 patent, 3:46–50.) That is, the CLI parser can translate CLI output into XML.

The ’886 patent further discloses embodiments of these operations with respect to Figure 2 (reproduced below).



In this exemplary method, input data is received at the IOS/CLI Parser (step 210). ('886 patent, 4:4–6.) The input data may originate from outside the system and may be in an XML format in “accordance with a specific XML schema of the CLI syntax.” ('886 patent, 4:15–16.) For example, the input data may be a request for the router to perform an operation. The input data can also originate from inside the routing system in “the form of CLI statements.” ('886 patent, 4:8–10.) When the input data originates from the router, it is generated by the router (*i.e.*, it is CLI output.) For example, the router may generate “input data” (*i.e.*, an output message) in response to executing a command. Once it is determined whether the input originates outside or inside the router, the input is traversed. ('886 patent, 4:17–24.)

Step 230 addresses XML-formatted input data received from outside the routing system. ('886 patent, 4:24–26.) In order to perform the request specified by the XML-formatted input data, the XML-formatted input is first translated into CLI statements. This translation occurs by extracting CLI command keywords and parameters from the XML according to the XML schema. ('886 patent, 5:23–36.) For example, the XML may be traversed and each tag may be converted to one or more corresponding CLI keywords. ('886 patent, 5:29–36.) Once the XML-formatted input is translated into CLI statements, the CLI statements are then executed by “Command Module 130 within routing system 100.” ('886 patent,

4:44–47.) In other words, even though the originating input data is XML-formatted, it is still eventually executed by the Command Module 130 as a normal CLI command.

Step 235 addresses input data originating from inside routing system 100 (*i.e.*, CLI output). In this case, the input data is in CLI format because it originated from the routing system. For example, it may be the output results from executing the received request. The translation of this data occurs by translating the CLI statements “in accordance with an XML schema of the CLI rules and behaviors.” (’886 patent, 4:36–39.) For example, tokens may be extracted from the CLI statement and rules may be applied to each token to convert it to XML. Table 3 (reproduced below) specifies embodiments of rules which may be applied to parse CLI tokens to XML:

TABLE 3

```

Open XML Tag and Closed XML tags + Value Rule
Open XML Tag is introduced by:
  Non-boolean value keyword parse node
  Keyword or parameter node's parent_label
For Parameter node:
  Add "<" to XML buffer
  Add parameter node's label string to XML buffer
  Add ">" to XML buffer
  Add parameter node's value to XML buffer
  Add "</" to XML buffer
  Add parameter node's label string to XML buffer
  Add ">" to XML buffer
For boolean keyword node (is_boolean is true)
  Add "<" to XML buffer
  Add keyword string to XML buffer
  Add ">true</" to XML buffer
  Add keyword string to XML buffer
  Add ">" to XML buffer
For non-boolean keyword node
  if (last character in buffer is not "_") {
    Add "_" to XML buffer
  } else (keyword's has_more_tag is false) {
    Add ">" to XML buffer
  }

```

Then, the translated input is passed along to the device that requested it (step 240). ('886 patent, 4:47–49.) In this step, an XML response is sent back to the original requestor.

III. Claim construction.

Petitioner asks the Board to construe the terms “command line interface (CLI) parser” and “parsing the output message to identify at least one CLI token.” The Board should decline to construe these terms because construction of these terms is not necessary to make a decision in this proceeding. *See e.g., Wangs*

Alliance Corp. v. Koninklijke Philips N.V., IPR2015-01287, Paper 8 at 5–6 (Nov. 25, 2015); *Johnson Controls, Inc. v. Wildcat Licensing WI, LLC*, IPR2014-00305, Paper 45 at 19 (June 22, 2015); *Motivepower, Inc. v. Cutsforth, Inc.*, IPR2013-00267, Paper 10 at 12 (Nov. 1, 2013).

IV. Ground 1: Petitioner failed to show that claims 1–10 are rendered obvious by Gorthy, Froyd, Courtney, and JUNOScript.

Petitioner applies four references in the single ground of unpatentability asserted in its Petition. But, none of the references asserted by the Petitioner, either alone or in combination, teach or suggest all of the elements of the challenged independent claims. As explained below, none of the references ever teach or suggest any conversion of CLI output messages that are generated based on executing a CLI command requested by an XML-formatted input command, which both challenged independent claims 1 and 6 require.

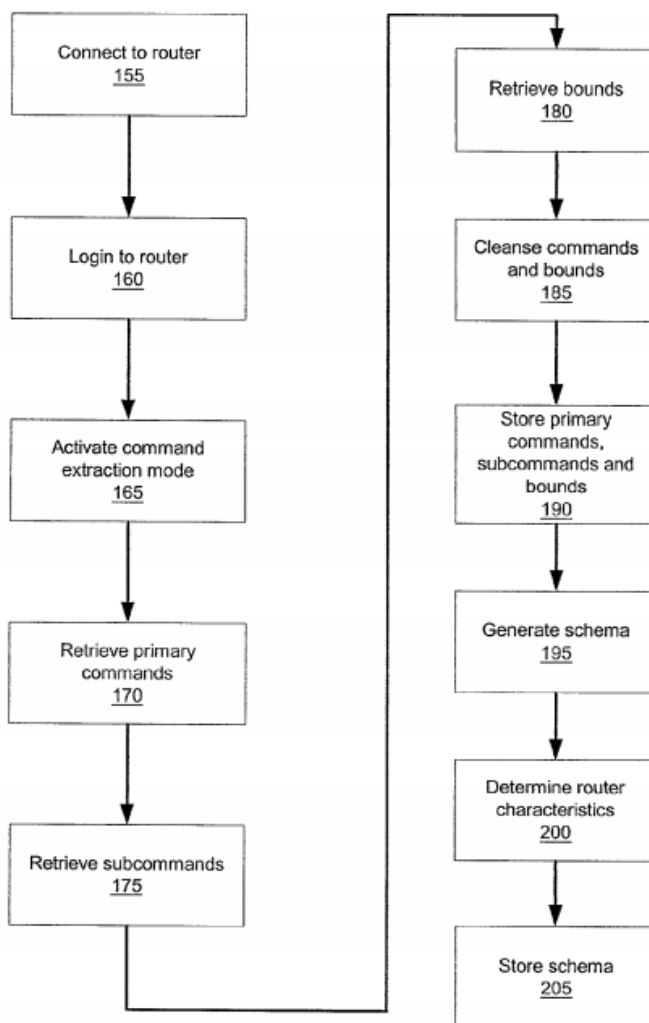
A. Overview of the applied references.

1. Gorthy's configuration schema-based system.

In contrast to the '886 patent, Gorthy describes a system that only addresses router commands (*i.e.*, input), not the output from a router's CLI. Gorthy's focus on input is understandable because Gorthy is directed to a different problem than the '886 patent—namely generating a model of the types of commands a router can receive. Gorthy performs this modeling process by generating configuration schemas for routers. (Exhibit 1003, Gorthy, 1:27–31.) Once a configuration

schema has been generated, the schema can then be used to convert between XML and router CLI configuration commands. (Gorthy, 5:9–10.)

Gorthy, however, does not disclose receiving any output from a router's CLI, and thus is not concerned with converting such output to XML. As Gorthy explains, routers can be difficult to configure because of the large number of commands and subcommands that the CLI allows. (Gorthy, 1:53–55.) Moreover, the particular commands that a router CLI can accept varies among routers and router software versions. (*Id.*) To solve these purported problems, Gorthy creates a single XML configuration schema that **models** the commands a router can receive. (Gorthy, 2:61–62.) Once this XML schema is created, it can be used to convert from XML-formatted commands to CLI-formatted commands. (Gorthy, 3:1–4.) This process is further discussed below with reference to Gorthy Figure 3:



As shown above, after connecting, and logging into a router, “command extraction mode” can be activated. (Gorthy, 4:46–48.) For example, on a Cisco router, “the command extraction mode is activating by entering a ‘?’ at the [CLI] prompt.” (Gorthy, 4:48–50.) If an administrator, enters “?” at the CLI prompt, the router will return all available commands that can be entered at the prompt, such as:

```

> ?
commands [ router
           admin
           global
           > router?
subcommands [ bgp
              ospf
              >

```

(Gorthy, 4:56–63 (with annotations).) If an administrator types a “?” after a command, each of the subcommands or bounds available for that command would be displayed. Thus, by repeatedly using the “?” command after all commands and subcommands, all commands, subcommands, and bounds can be discovered and retrieved from a router. (Gorthy, 4:64–65.)

Once the commands, subcommands, and bounds are collected, Gorthy describes that this data can be used to “build a configuration schema, which in essence is a modeling of the router’s command structure (step 195).” (Gorthy, 5:9–10.) This configuration schema is then “associated with [the] characteristics of the router.” (Gorthy, 5:50–53.) For example, “the configuration schema might be associated with a Cisco™ router, model 7500, OS version 12.0.” (Gorthy, 5:50–54.) Thus, Gorthy does not teach converting any CLI output to XML, but rather only modeling the commands a router can accept.

Gorthy’s configuration schema can then be used, such that “a system administrator 125 can reconfigure such a router using XML-based commands....”

(Gorthy, 6:3–7.) For example, a system administrator can send an XML-based command to Gorthy’s configuration interface, where it would then be converted to “a CLI-based command using the XML schema.” (Gorthy, 6:11–12.) Then, the CLI-based command can “be passed to the configuration storage module 145 where it is integrated into the configuration of the router.” (Gorthy, 6:13–15.) It should also be noted that the XML-to-CLI conversion in Gorthy is done by the converter 235 (*id.*), and not by the CLI parser as required in claim 1 of the ‘886 patent.

2. Froyd’s generalized markup.

Froyd is generally directed to a system for “storing and restoring system configuration using generalized markup language.” (Exhibit 1004, Froyd, 1:13–15.) Like Gorthy, Froyd is not concerned with the output of router CLIs. Instead, Froyd recognizes that devices may have a wide variety of management interfaces. (Froyd, 1:19–24.) And the way a user monitors or configures a device may vary widely depending on the device. (Froyd, 1:36–66.) This may be complex to manage, so Froyd describes a “generalized markup language” that can be used to describe the configuration of a device in a consistent manner across devices that have different interfaces. (Froyd, 2:11–3.) This generalized markup language can be in the form of XML. (Froyd, 11:16–17.) The XML can “be used to create a new set of commands that describe commands associated with a Lucent router such that

the new set of command[s] is consistent with commands associated with routers from Cisco.” (Froyd, 11:20–24.) Moreover, the generalized XML could “shield the users from changes or version upgrades applied to the operating software of the system.” (Froyd, 11:24–28.) In other words, the generalized XML can remain consistent even across different versions of a system.

According to Froyd, with respect to CLI commands, the generalized XML uses “the <command> tag ... for a single command.” (Froyd, 8:51–52.) The command tag does not have any XML attributes and instead includes a “single <keyword> tag.” (Froyd, 8:51–54.) The following generalized XML defines the “show” command:

```
<command>  
    <keyword text="show">  
    </keyword>  
</command>.
```

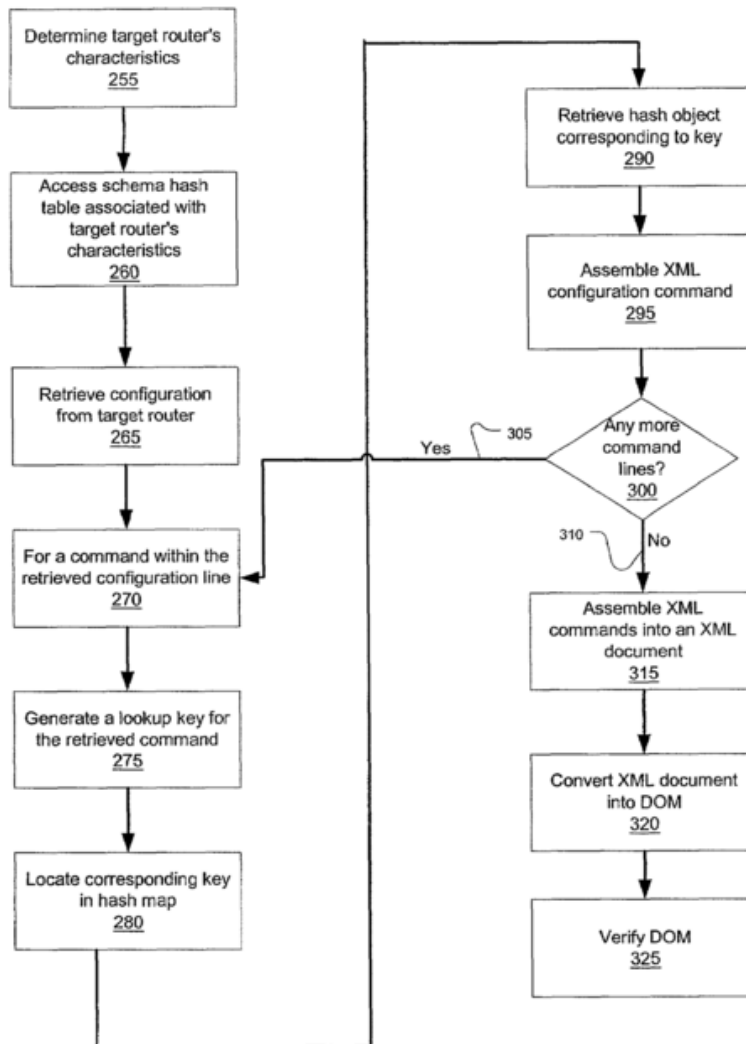
(Froyd, 8:58–61.)

3. Courtney’s network device configuration modeling system.

Courtney solves a different problem than the ’886 patent—modeling the configuration of a network device. (Exhibit 1002, Courtney, abstract) Unlike the ’886 patent, Courtney does not describe a CLI that can communicate in XML. Rather, Courtney is only concerned with the creation of an XML model of a router’s configuration. Although Courtney recognizes that its schemas could be

used to convert XML commands to CLI commands, Courtney (like Gorthy) does not receive any output from a CLI, and thus cannot convert such CLI output to XML. Courtney explains that a router's CLI can be difficult to use and XML is more "user friendly." (Courtney, 1:51.) Courtney retrieves a "network device's configuration, in a native format, from the network device ... and converting it into a standard-format configuration such as an XML document or DOM." (Courtney, 2:40–45.) Courtney does not disclose how the retrieval of the network device's configuration is accomplished. Courtney simply states that the conversion occurs by using a "schema [that] can be directly used to generate an XML document that represents the configuration of the particular network device." (Courtney, 3:12–14.) In Courtney, the particular schemas used for the conversion depend on the particular type of network device, and these schemas are previously generated. (Courtney, 2:65–3:1.) Courtney itself, however, does not describe how these schemas are created, other than stating that a "CLI-to-XML" converter (*e.g.*, by using command extraction mode), such as described in Gorthy, can be used.

In the preferred embodiment, Courtney discloses that an "intermediate representation (*e.g.*, a hash representation), of the schema is generated and the intermediate representation is used to more quickly generate the corresponding XML document." (Courtney, 3:15–18.) This process is further illustrated below in Figure 6 of Courtney:



As shown, once the router's configuration is retrieved, for each native command in the router's configuration, a lookup key is generated for the command. The lookup key is used as an index into a "hash table." (Courtney, 7:26–31.) For each lookup key, the hash table stores a hash map object that "contains schema information regarding the command or value such as whether optional or required data type, etc." (Courtney, 7:31–33.) Using this information

from the hash map object, Courtney's XML converter can assemble the XML-based command that corresponds to the router's native command (step 295). (Courtney, 7:33–35.)

4. JUNOScript's router API.

JUNOScript is a manual that “describes how to use [Release 5.0 of] the JUNOScript application programming interface (API) to configure or request information from the JUNOScript server running on a Juniper Networks router.” (Exhibit 1005, JUNOScript, p. 9.) Specifically, the JUNOScript API is a set of XML “tags that describe router components” for “Juniper Networks customers who want to write custom applications for router configuration or monitoring.” (*Id.*) Although JUNOScript describes that its interface can both receive XML commands and send XML output, JUNOScript does not convert XML input to CLI commands or convert CLI output to XML.

JUNOScript explains that the reason it does not convert between CLI and XML is that it is too difficult due to the unpredictability in a router's CLI. (JUNOScript, p. 17.) Therefore, JUNOScript skips sending any XML commands to the router's CLI, and does not convert from XML to CLI format or vice versa. Rather, all of the XML commands received at a JUNOScript router are handled directly by the router's internal software modules, and the output is generated directly by those modules in XML. (JUNOScript, p. 15) Specifically “[c]lient

applications can configure or request information from a router by encoding the request with JUNOScript tags and sending it to the JUNOScript server running on the router.” (JUNOScript, p. 15.) The JUNOScript server does not send JUNOScript requests to the JUNO CLI, and instead the server sends those requests directly to the internal software of the router. When the JUNOScript server, located on the router, receives the response, it encodes the response in JUNOScript tags, and returns the result to the client application. (*Id.*) For example, if a client application makes the request encoded with JUNOScript tags:

Client Application

```
<rpc>
  <get-chassis-inventory>
    <detail/>
  </get-chassis-inventory>
</rpc>
```

(JUNOScript, p. 38.). The JUNOScript server may respond with the following JUNOScript encoded response without the use of any CLI:

```
<rpc-reply>
  <chassis-inventory>
    <chassis>
      <name>Chassis</name>
      <serial-number>1122</serial-number>
      <description>M10</description>
      <chassis-module>
        <name>Midplane</name>
        <!-- other child tags for the Midplane chassis module -->
      </chassis-module>
      <!-- tags for other chassis modules -->
    </chassis>
  </chassis-inventory>
</rpc-reply>
```

(*Id.*)

JUNOScript also describes being able to display JUNOScript within the JUNOS CLI, although it does not provide specifics for how this is accomplished, nor is such output generated in response to an XML command. JUNOScript explains that “[t]o display the output from a JUNOS CLI command as JUNOScript tags rather than the default formatted ASCII, pipe the command to the display xml command.” (JUNOScript, p. 37.) For example, as shown below, JUNOScript describes that if one executes the commands “show chassis hardware | display xml” in the JUNOS CLI, the following JUNOScript output is displayed:

```
user@host> show chassis hardware | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/5.1R1/dtd/junos.dtd">
<chassis-inventory xmlns="http://xml.juniper.net/junos/5.1R1/dtd/junos-chassis.dtd">
<chassis junos:style="inventory">
<name>Chassis</name>
<serial-number>00118</serial-number>
<description>M40</description>
<chassis-module>
<name>Backplane</name>
<version>REV 06</version>
<part-number>710-000073</part-number>
<serial-number>AA2049</serial-number>
</chassis-module>
<chassis-module>
<name>Power Supply A</name>
...
</chassis-module>
...
</chassis>
</chassis-inventory>
</rpc-reply>
```

(JUNOScript, p. 37.)

B. Petitioner failed to show that the combination of Gorthy, Froyd, Courtney, and JUNOScript discloses the “translating an output message...” element of the challenged claims.

Each of the challenged independent claims recites “translat[ing] an output message, generated in response to the performance of the operation, from a CLI format to an XML format having the CLI syntax.” (’886 patent, 7:59–61; 8:58–59.) “The operation” recited in this claim element is the operation requested by the original XML-formatted input command:

receiving, with a command line interface (CLI) parser, an input command configured to request **an operation** be performed by a routing system, ... the input command from the XML format having the CLI syntax into a CLI command that, when executed, is configured to prompt the routing system to perform **the operation**,

...

wherein the routing system is configured to perform **the operation** responsive to the execution of the CLI command;

translating an output message, generated in response to performance of **the operation** ...

(’886 patent, claim 1, 7:38–60 (with annotations).) Thus, the output message translated in this claim element results from the performance of the operation requested in the original XML-formatted input message. To establish obviousness of this claim element, Petitioner was therefore required to show the alleged

translated “output message” is tied to the performance of an operation requested in the alleged XML-formatted input message.

Petitioner raises three independent arguments attempting to demonstrate that its proposed combination teaches or suggests this claim element. First, Petitioner argues that Courtney discloses this element by translating “the current configuration of a system” to XML. (Petition, pp. 36–37.) Second, Petitioner argues that JUNOScript discloses the claimed “translating an output message” because JUNOScript discloses transmitting XML as part of its application programming interface (API). (Petition, pp. 38–39.) Third, Petitioner argues that JUNOScript teaches “translating an output message” because it includes “pipe” and “display XML” commands that when used together output XML in its CLI. (Petition, p. 39.) Each of Petitioner’s arguments fails because none of its alleged output messages result from the performance of an operation requested by a translated XML input message.

1. Courtney’s XML router configuration is not the claimed “output message.”

Petitioner first argues that Courtney discloses the output translation claim element because “Courtney discloses a converter that translates an output message in a CLI format to an XML format.” (Petition, p. 36.) Petitioner then speculates that “one sort of output message that can be translated is an output message

providing the current configuration of the system.” (Petition, p. 36.) However, Petitioner faces a conundrum. The claims require a specific type of output message be translated—an output message generated in response to a requested operation in an XML-formatted input command. But, Courtney’s router configuration is not retrieved in response to executing any command, let alone an XML-formatted input command message. Indeed, Courtney explicitly states that the router configuration is retrieved directly “from the configuration storage module” in its “native format” not in response to a CLI command. (Courtney, 2:40–45.)

Petitioner therefore turns to Gorthy arguing that the current configuration of a system “can be done through what Gorthy refers to as ‘a command extraction mode,’ which ‘is activat[ed] by entering a “?” at the prompt.’” (Petition, p. 36.) First, this disclosure in Gorthy, combined with Courtney, does not disclose the limitations of the claims of the ’886 patent because “?” is not an input command configured in an extensible markup language, nor is there any disclosure in Gorthy of an XML-formatted command that translates to the “?” CLI command. Further, Petitioner misunderstands Gorthy’s “command extraction mode.” Gorthy’s “command extraction mode” is not capable of retrieving the configuration of a router. Instead, Gorthy’s “command extraction mode” retrieves the “primary commands, subcommands and bounds” a router is capable of executing. In other

words, Gorthy's "command extraction mode" retrieves the commands an administrator can execute in the CLI of a router, not the configuration of the router.

In addition to Gorthy's explicit teachings that its "command extraction mode" is used to retrieve "commands, subcommands and bounds" and not router configuration, Gorthy's method of performing "command extraction" uses the well-known router "help" command, which a person of ordinary skill in the art would recognize is not used for retrieving router configuration. Specifically, Gorthy performs "command extraction" using the "?" on a Cisco router CLI. (Exhibit 2001, Cisco Networking for Dummies, p. 302.) A person of ordinary skill in the art would have understood that on a Cisco router, the "?" command is a "context-sensitive help" command that "tries to guess what you are trying to do and provides you with help..." (*Id.*) Specifically, the help command operates in two modes: "word help" and "command syntax help." (Cisco Networking for Dummies, p. 303.)

In "word help" mode, the "?" command "displays a list of the available commands" in that context. (*Id.*) For example, as Gorthy explains, if a user enters "?" at the first prompt in a CLI, the router would display the available commands of "router," "admin," and "global." (Gorthy, 4:56–59.) In "command syntax help" mode "IOS will provide you with the command structure and parameter list if you forget." For example, as Gorthy explains, if a user enters "router ?," the CLI would

display that the possible parameters of the “router” command are “bgp” and “ospf.” (Gorthy, 4:60–63.) Thus, when Gorthy describes that “this process could be repeated until termination for each command and subcommand” it means that one could use a combination of “?” by itself and commands intermixed with a “?” to extract all possible commands and subcommands from a router. (Gorthy, 4:64–66.) And, the universe of commands a router can accept is plainly not the same as the configuration of the router. For example, in Appendix A, Gorthy describes the universe of “service” commands a router can accept. (Gorthy, Appendix A.) One command listed “service alignment logging \$Enable logging of alignment issues\$.” (Gorthy, 7:3.) But, the “service alignment logging” command, in and of itself, is not the configuration of a router. Rather, it is merely a command that an administrator can enter into the CLI. Perhaps the “service alignment logging” command, **when executed**, may trigger a change to the configuration of the router, but merely the fact that a router can receive this command is not the configuration of a router.

Thus, a person of ordinary skill in the art would not use Gorthy’s “command extraction” mode to retrieve the router configuration as described in Courtney. And, accordingly, Petitioner’s first argument fails to demonstrate that Courtney, alone or in combination with Gorthy, Froyd, and JUNOScript teaches or suggests the output message translation element of the independent claims.

2. JUNOScript does not teach or suggest the claimed “output message.”

Petitioner next argues that JUNOScript discloses the claimed output translation element based on two alternative theories. (Petition, pp. 38-41.) First, Petitioner argues that JUNOScript’s XML API translates CLI output into XML output. (Petition, pp. 38-39.) But, JUNOScript explicitly disparages translating CLI output, and Petitioner does not point to any explicit teachings of JUNOScript that describe such translation. Second, Petitioner argues that JUNOScript’s “pipe” function combined with the “display xml” command performs the claimed output translation. (Petition, pp. 39-41.) But, Petitioner’s argument fails because JUNOScript’s “pipe” function and “display xml” commands are not performed in response to receiving an XML-formatted input command—they are not capable of being executed via JUNOScript’s XML interface. Moreover, JUNOScript’s “pipe” function and “display xml” command do not perform translation between CLI output and XML “according to a stored mapping of CLI tokens-to-XML values” because the generated XML is retrieved directly from the router’s configuration database, not from translating CLI output.

a) JUNOScript’s XML output is not “generated in response to the operation.”

Petitioner alleges that JUNOScript “provides another example in the prior art of translating each CLI token of the output message into a corresponding XML

value according to a stored mapping of CLI tokens-XML-value.” (Petition, pp. 38–39.) Specifically, Petitioner points to the following exemplary output message from the JUNOScript reference:

```
Physical interface: fxp0, Enabled, Physical link is Up  
Interface index: 4, SNMP ifIndex: 3
```

(JUNOScript, p. 17.) Petitioner then states that JUNOScript’s API translates this output message into the following XML message:

```
<interface>  
  <name>fxp0</name>  
  <admin-status>enabled</admin-status>  
  <operational-status>up</operational-status>  
  <index>4</index>  
  <snmp-index>3</snmp-index>  
</interface>
```

(*Id.*)

From these two examples, without a single citation to JUNOScript or any other reference, Petitioner concludes that JUNOScript “parses the output message shown above [i.e., the CLI output] to identify at least one CLI token and translates each token to a corresponding XML value,” thus generating the above XML output. (Petition, p. 39.) Petitioner’s unsupported conclusions should be disregarded. First, JUNOScript never discloses that the XML generated in its API is generated by executing a CLI command, or that the XML shown above was generated by parsing the CLI output. Second, as explained above in the overview

of JUNOScript, JUNOScript teaches avoiding parsing CLI output due to the unpredictability of router's CLIs. (JUNOScript, p. 17.)

The only evidence provided by Petitioner is a citation to a single paragraph of Dr. Clark's declaration. But, Petitioner's citation to its expert declaration does not save its argument because Dr. Clark provides no expert insights, reasoning or analysis of why or how JUNOScript performs such "parsing" or "translating," merely parroting the same speculations that appear in the Petition. (*See* Clark Decl., ¶ 80.) "Merely repeating an argument from the Petition in the declaration of a proposed expert does not give that argument enhanced probative value," and a petitioner cannot move forward to trial based upon such "mere conclusory statements." *See, e.g., Kinetic Techs., Inc. v. Skyworks Solutions, Inc.*, IPR2014-00529, Paper. 8 at 15–16 (Sept. 23, 2014) (citing *In re Kahn*, 441 F.3d 977, 988 (Fed. Cir. 2006)).

Thus, Petitioner's argument fails to demonstrate that JUNOScript alone or in combination with Gorthy, Courtney, and Froyd teaches or suggests the output message translation element of the independent claims.

b) JUNOScript's "pipe" function combined with the "display xml" command does not teach or suggest the output translation claim element.

Petitioner also argues that the use of "the 'pipe' function and the 'display xml' command" in the JUNOScript API "translates an output message, generated

in response to the performance of an operation, from a CLI format into an XML format having a CLI syntax.” (Petition, p. 39.) Petitioner’s argument fails for two fundamental reasons. First, Petitioner’s cursory handling fails to establish that the “pipe” function and the “display xml” command of JUNOScript alone or in combination with any of the three other applied references generates and translates an output message in response to an operation requested in an **XML-formatted** input command. Second, Petitioner fails to establish that the alleged “translation” associated with the “display xml” command is according “to a stored mapping of CLI tokens-to XML values.”

(1) JUNOScript’s “pipe” function combined with the “display xml” command is not an output message in response to an operation requested in an XML-formatted input.

Petitioner relies on JUNOScript’s disclosure of a “pipe” function combined with the “display xml” command as the claimed “output message” translation element of both independent claims. However, Petitioner never explains exactly how it proposes to combine the “| display XML” feature of JUNOScript with the system resulting from the combination of Gorthy, Courtney and Froyd. Petitioner’s high-level discussion of the motivation to combine provides some insights:

Both Gorthy and Courtney discuss the XML-based interface of Juniper Networks routers ... Those references praise of such XML-based interfaces would have directed a person of ordinary

skill in the art to documentation regarding Juniper Networks routers, such as JUNOScript Guide. Clark Decl. ¶54. Moreover, these references are directed to the same field of endeavor—frameworks for configuring networking equipment using XML—and teach similar methods of using XML-based commands with a CLI syntax.

(Petition, p. 24.) Based on this discussion and the testimony of Dr. Clark, who merely repeats Petition statements verbatim, Petitioner may be proposing to use the teachings of JUNOScript to modify the XML-based interface of the combined system of Gorthy, Courtney, and Froyd. However, the combination does not result in the claimed system recited in independent claims 1 and 6, which each require that the output message be in response to an operation requested in an **XML-formatted** input command.

Even a cursory review of JUNOScript's guide shows the flaw in Petitioner's argument—the pipe function and the display xml command (“| display XML”) are entered in a CLI format directly into the CLI, and **not** entered in an XML-formatted input command via an XML API. For example, the figure below illustrates that a user enters the input command “show chassis hardware | display xml” into the CLI in a CLI format (not an XML format).

The diagram illustrates a command-line interface (CLI) interaction. On the left, an arrow labeled "input" points to the command `user@host> show chassis hardware | display xml`. Another arrow labeled "output" points to the resulting XML response. The XML is an RPC reply containing chassis inventory data.

```

user@host> show chassis hardware | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/5.1R1/dtd/junos.dtd">
  <chassis-inventory xmlns="http://xml.juniper.net/junos/5.1R1/dtd/junos-chassis.dtd">
    <chassis junos:style="inventory">
      <name>Chassis</name>
      <serial-number>00118</serial-number>
      <description>M40</description>
      <chassis-module>
        <name>Backplane</name>
        <version>REV 06</version>
        <part-number>710-000073</part-number>
        <serial-number>AA2049</serial-number>
      </chassis-module>
      <chassis-module>
        <name>Power Supply A</name>|
      ...
    </chassis-module>
    ...
  </chassis>
</chassis-inventory>
</rpc-reply>

```

(JUNOScript, p. 37 (annotations and emphasis added).) In this example, the “show chassis hardware” command is “piped” to the “display xml” command, causing the output to be presented to the user in XML format, as shown above. But, the “show chassis hardware | display xml” input command is not an XML formatted input command. In fact, it is not possible to send the “display xml” command to a JUNOScript router via its XML API. For example, the JUNOScript API Reference, describing all of the commands that the JUNOScript API is capable of receiving, does not list any command that would execute “display xml.” (See e.g., Exhibit 2004, JUNOScript API Reference, pp. 15–18.) Thus, the only possible way to execute the “pipe” function and “display xml” command would be through JUNOScript router’s CLI which would preclude the resulting output message from being created in response to an input command formatted in XML, as required by both challenged independent claims 1 and 6.

(2) *JUNOScript’s “pipe” function combined with the “display xml” command does not translate its output into XML according to a stored mapping of CLI tokens-to-XML values.*

Petitioner also fails to establish that JUNOScript discloses translating its output into XML “according to a stored mapping of CLI tokens-to-XML values” as required by both challenged independent claims. To support its position, Petitioner speculates that “[b]y using the “| display xml” feature, the JUNOScript API translates the default CLI output to XML.” (Petition, p. 40.) However, this speculation is premised on a misunderstanding of JUNOScript and Juniper routers, which is fatal to Petitioner’s argument.

Petitioner contends that by piping the output of a command to the “display XML” command, JUNOScript translates the default CLI text into XML. Petitioner assumes, as they must, that a “pipe” causes the “CLI formatted output messages” to be “the input to the ‘display XML’ function,” which in turn generates an XML formatted output message. Petitioner builds on this faulty foundation adding that “it is both inherent and obvious that the JUNOScript API translates the output message ‘according to a stored mapping of CLI tokens-to-XML values,’ as there would be **no other way** for the software to produce the XML output from a pipe (“|”) function, which uses the default CLI command output as the input for the ‘display xml’ function.” (Petition, p. 40 (emphasis added).)

Petitioner cites no evidence for its position other than a paragraph from its expert declaration. But, Dr. Clark's opinion does not salvage Petitioner's argument because, once again, it only repeats verbatim the same statements as the Petition with no additional explanation or support. (Clark, ¶ 82.) *See also, Kinetic Techs.* at 15–16 (finding that a declaration that repeats the argument from the Petition does not give that argument enhanced “probative value”). This lack of supporting evidence is not surprising because basic research into the “display xml” function demonstrates that Petitioner's understandings are incorrect. The “display xml” does not even translate to XML, let alone translate to XML according to a stored mapping of CLI tokens-to-XML values.

First, contrary to Petitioner's unsupported assumption, the “display xml” command does not use the output of the piped command to form its XML. Instead, it retrieves its XML directly from the Juniper router's configuration database. (Exhibit 2002, Juniper MX Series, p. 7.) Whenever a user interacts in the CLI of a Juniper router, the interaction actually occurs with a process known as the “management process” or “MGD.” (Exhibit 2003, JUNOS OS for Dummies, p. 352.) “**All** communications” with the MGD process “are actually taking place using XML.” (*Id.*) For ease of display, the MGD process “strips off the XML tags before sending the output to your screen.” (*Id.*) That is, when JUNOScript displays textual output from a CLI command, it is actually converted from XML. But, when

the “display xml” command is used, MGD reverts to its native XML language, which is received from “MGD via RPC.” (Juniper MX Series, p. 7.) Thus, Petitioner is incorrect that piping a command output to “display xml” results in a translation from ASCII to XML because a Juniper router’s native output is XML.

Thus, Petitioner’s final argument fails to demonstrate that JUNOScript alone or in combination with Gorthy, Courtney, and Froyd discloses the output translation element of the independent claims.

C. Petitioner has failed to establish that a person of ordinary skill in the art would combine Gorthy, Courtney, Froyd, and JUNOScript.

Petitioner cites four references in its single proposed ground of patentability—Gorthy, Courtney, Froyd, and JUNOScript. Petitioner spends four pages of its Petition purportedly setting out a motivation to combine these four references. (Petition, pp. 22–25.) But, nowhere in these pages does Petitioner explain how these references would be combined to achieve the claimed invention. Indeed, Petitioner’s motivation to combine discussion occurs before any discussion of these references relative to the claims. Moreover, when Petitioner does present its obviousness arguments, Petitioner addresses each claim element independently without establishing that the invention as a whole was rendered obvious by its combination. Patent claims are “not proved obvious merely by demonstrating that each of [their] elements was, independently, known in the prior art.” *KSR Int’l v.*

Teleflex Inc., 550 U.S. 398, 418 (2007). “Rather, obviousness requires the additional showing that a person of ordinary skill at the time of the invention would have selected and combined those prior art elements in the normal course of research and development to yield the claimed invention.” *Unigene Labs., Inc. v. Apotex, Inc.*, 655 F.3d 1352, 1360 (Fed. Cir. 2011).

Petitioner fails to make the requisite showing for obviousness because Petitioner never explains that a person of skill in the art would have selected and combined the identified elements of the cited references to yield the claimed invention. *Medtronic, Inc. v. Robert Bosch Healthcare Sys.*, Case IPR2014-00436, Paper 17 at 12–15 (June 19, 2014) (denying institution because of lack of showing of reasonable rationale for a combination); *TRW Auto US LLC v. Magna Elecs. Inc.*, Case IPR2014-00296, Paper 15 at 15–16 (July 3, 2014) (denying institution because motivation to combine was insufficient); *Bumble Bee Foods, LLC v. Kowalski*, Case IPR2014-00224, Paper 18 at 18 (June 5, 2014) (same); *Google Inc. v. EveryMD.com LLC*, IPR2014-00347, Paper 9 at 25–27 (May 22, 2014) (same); *General Electric Co. v. TAS Energy Inc.*, IPR2014-00163, Paper 11 at 16–20 (May 13, 2014); *ManukaMed Ltd. v. Apimed Medical Honey Ltd.*, IPR2013-00234, Paper 17 at 20–22 (Sept. 25, 2013) (same).

Here, Petitioner has done nothing more than assert that a person of ordinary skill in the art would have combined Gorthy, Froyd, Courtney, and JUNOScript

because each of the four references, either relates to a “translation to and from XML,” “routers,” or “frameworks for configuring networking equipment using XML.” (Petition, pp. 22–25.) But, merely because these references may describe technologies in the general field of using CLI and XML with network equipment is not a sufficient motivation to combine. An assertion of obviousness “cannot be sustained by mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness.” *Id.* (citing *In re Kahn*, 441 F.3d 977, 988 (Fed. Cir. 2006)).

With respect to Froyd, Gorthy, and Courtney, Petitioner states that “one of skill in the art would have been motivated to combine Froyd with either or both Courtney and Gorthy because each is concerned with the use of translations to and from XML in order to provide users with an easy to use standard format.” (Petition, p. 23.) But, even assuming that Froyd, Gorthy and Courtney do each describe “translations to and from XML,” Petitioner still has not met its burden of explaining why a person of ordinary skill in the art would apply the teachings of Froyd to Gorthy and Courtney. Simply describing the same system is not enough—Petitioner needs to show why a person of skill in the art would choose the particular teachings Petitioner is relying on from Froyd to combine with Gorthy and Courtney. *Unigene Labs.*, 655 F.3d at 1360. Petitioner fails to address this requirement of obviousness. Petitioner’s expert, Dr. Clark, does not support

Petitioner’s argument—far from it—because the supporting paragraphs from Dr. Clark’s declaration merely repeats Petitioner’s arguments, providing no expert knowledge, insights, or analysis. (*Compare* Clark Dec., ¶¶ 50–55 *with* Petition pp. 22–25.) *See also*, *Kinetic Techs.*, at 15–16 (finding that a declaration that repeats the argument from the Petition does not give that argument enhanced “probative value”).

Petitioner also fails to provide a sufficient motivation to combine JUNOScript with Gorthy, Froyd, and Courtney. Petitioner contends that a person of ordinary skill in the art would combine JUNOScript with the other references because each of the references “relate[] to using XML-tagged commands to configure and request information from routers,” and that the “inventions complement each other.” (Petition, p. 24.) But Petitioner’s analysis ends there. Petitioner never explains why the “inventions complement each other” or whether or not the references “relate to using XML-tagged commands to configure and request information from routers” would motivate a person of skill in the art to combine them together. Moreover, as explained above, JUNOScript teaches not converting from CLI to XML, yet Petitioner never explains why a person of ordinary skill in the art would rely on JUNOScript to convert from CLI to XML despite its contrary teachings. And, just as above, Dr. Clark is of little help because his opinion mirrors the words of the Petition nearly verbatim. (Clark Decl., ¶¶ 54–

55.) *See also, Kinetic Techs.*, at 15–16 (finding that a declaration that repeats the argument from the Petition does not give that argument enhanced “probative value”).

Accordingly, Petitioner has not met its burden for establishing that a person of ordinary skill in the art would have combined Gorthy, Froyd, Courtney, and JUNOScript.

D. Petitioner did not establish that JUNOScript is a § 102 prior art printed publication.

Petitioner argues that JUNOScript qualifies as a prior art publication under § 102(b). (Petition, p. 21.) Because Petitioner did not offer competent evidence to show that JUNOScript was publically available before the earliest priority date of the '886 patent, July 8, 2005, Petitioner has not established that JUNOScript is a prior art printed publication eligible for this IPR proceeding.

To show that JUNOScript is a prior art printed publication, Petitioner must demonstrate that interested persons of ordinary skill in the art, exercising reasonable diligence, could have located it before the '886 patent priority date.

Blue Calypso LLC v. Groupon Inc., Case. No., 2015-1396, slip op. at pp. 28–29 (Fed Cir., March 1, 2016); *see also SRI Int’l v. Internet Sec. Sys.*, 511 F. 3d 1186, 1194–95 (Fed. Cir. 2008). Petitioner alleges that JUNOScript was published on

November 6, 2001. (Petition, p. 21.) But, Petitioner provides no evidence establishing that JUNOScript was publically available on that date.

Presumably, Petitioner believes JUNOScript was published on November 6, 2001 because the JUNOScript document states that it was revised on “6 November 2001” and has a copyright date of “2001.” (JUNOScript, p. 2.) But it is well settled that the mere appearance of dates in document, such as copyright dates or revision dates, do not establish that the document was available to and/or disseminated to the public. *See, e.g., In re Lister*, 583 F.3d 1307, 1316–17 (Fed. Cir. 2009) (finding that a manuscript registered with the copyright office was not prior art because there was insufficient evidence to show that the manuscript was publicly accessible prior to the critical date); *Mannesmann Demag Corp. v. Engineered Metal Products Co., Inc.*, 605 F. Supp. 1362, 1366–67 (D. Del. 1985), *aff’d*, 793 F.2d 1279 (Fed. Cir. 1986) (concluding that a patentee’s prior publication was not prior art under 102(b), “[a]bsent more, the mere fact that the Demag brochure was dated, ‘6.77,’ is hardly evidence that the brochure was actually disseminated to the relevant public as of that date”); *Neutrino Dev’t Corp. v. Sonosite Inc.*, 337 F.Supp.2d 942, 947 (S.D. Tex. 2004), *aff’d*, 210 Fed. Appx. 991 (Fed. Cir. 2006) (finding that a brochure affixed with the date April of 1996, at best, showed that it was printed in April of 1996, but insufficient to establish that it was a “printed publication”). Moreover, these dates are inadmissible hearsay. *Valeo Inc., et al. v.*

Magna Elec., Inc., IPR2014-00222, Paper 13 at 17 (Apr. 23, 2015); *see also* *Hilgraeve, Inc. v. Symantec Corp.*, 271 F.Supp.2d 964, 974 (E.D. Mich. 2003) (copyright “dates imprinted on ... documents are hearsay when offered to prove the truth of the matter asserted”).

Petitioner did not establish that JUNOScript was published before the earliest possible priority date of the ’886 patent and thus failed to establish that JUNOScript qualifies as prior art printed publication. For this additional reason, Petitioner failed to make its *prima facie* case of obviousness. Accordingly, the Board should deny Ground 1 as to all claims.

V. Conclusion.

Petitioner has not met its burden to prove that claims 1–10 are unpatentable. Petitioner relies on a single ground of unpatentability, combining four references. But, even having the benefit of the teachings of four separate references, Petitioner’s obviousness combination falls short. None of these references, alone or in combination, renders obvious any of the challenged claims of the ’886 patent. Accordingly, the Board should find claims 1–10 patentable over the proposed ground and deny Institution.

IPR2016-00244
U.S. Patent No. 7,953,886

Respectfully submitted,

STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.

/Lori A. Gordon/

Lori A. Gordon (Reg. No. 50,633)
Daniel S. Block (Reg. No. 68,395)
Attorneys for Patent Owner

Date: March 3, 2016
1100 New York Avenue, N.W.
Washington, D.C. 20005-3934
(202) 371-2600

CERTIFICATION OF SERVICE

The undersigned hereby certifies that the foregoing **PATENT OWNER PRELIMINARY RESPONSE**, Exhibit List, and all associated exhibits were served electronically via e-mail on March 3, 2016, in their entirety on the following:

Leo Lam (Lead Attorney)
Eugene M. Paige (Back-up Counsel)
Robert A. Van Nest (Back-up Counsel, pending filing of *pro hac vice* motion)
Brian L. Ferrall (Back-up Counsel, pending filing of *pro hac vice* motion)
David J. Silbert (Back-up Counsel, pending filing of *pro hac vice* motion)
Keker & Van Nest LLP
aristaipr@kvn.com
llam@kvn.com
dsilbert@kvn.com

STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.

/Lori A. Gordon/

Lori A. Gordon
Registration No. 50,633
Attorney for Patent Owner

Date: March 3, 2016

1100 New York Avenue, N.W.
Washington, D.C. 20005-3934
(202) 371-2600